mml-models meet data

Three major components of a machine learning system: data, models, and learning.

The *model* is obtained by *learning* from the *data.*

The *prediction* is made by applying a learned *model* on test *data.*

Data as vectors

For a supervised learning algorithm, there's both data points and labels, denote the dataset as $X = \{(x_1, y_1), \ldots, (x_N, y_D)\} \in \mathbb{R}^{N \times D}$.

- x_i: data point/example
- y_i: label
- N: numbers of data points/examples
- D: numbers of attributes/features

Model as functions

Model is a predictive function, known as a *predictor*, that, when given a particular input example/data point, produces an output.

Consider the special case of the linear functions:

$$f(x) = \theta^T x + \theta_0$$

Model as probability distributions

The observed data is a combination of true underlying data and noise.

> We want predictors that express some sort of uncertainty, e.g. to quantify the confidence we have about the value of the prediction for a particular test data point.

So consider predictors as probability models.

Learning is finding parameters

Goal of learning is to find a model and its corresponding parameters such that the resulting *predictor will perform well on unseen data.*

There are conceptually three distinct algorithmic phases when discussing machine learning algorithms:

Prediction or inference

is when using the trained predictor on the previously unseen test data.

Training or parameter estimation

is when adjusting predictive model based on training data, to find good predictors given training data.

To find good predictors:

- finding the best predictor based on some measure of quality
- using Bayesian inference, for probabilistic models

For **non-probabilistic** models, follow the *empirical risk minimisation*, which directly provides an optimisation problem for finding good parameters.

For **statistical** model, follow the *maximum likelihood*, which is used to find a good set of parameters.

Simulate the behaviour of the predictor on future unseen data using *cross-validation*

Regularisation: to balance between fitting well on training data and finding ‘simple’ explanations of the phenomenon

Hyperparameter tuning or model selection

Make high-level model decisions about the structure of the predictor, such as the class of probability distributions or

- hyperparameter: the number of components to use.
- model selection: the problem of choosing among different models

For non-probabilistic models, model selection is often done using nested cross-validation.

Hyperparameters: number of layers in deep learning, components in Gaussian Mixture model…

Parameters: numerically optimised

Remark. The distinction between parameters and hyperparameters is somewhat arbitrary, and is mostly driven by the distinction between what can be numerically optimized versus what needs to use search techniques. Another way to consider the distinction is to consider parameters as the explicit parameters of a probabilistic model, and to consider hyperparameters (higher-level parameters) as parameters that control the distribution of these explicit parameters. ♢

Empirical Risk Minimisation

Solve 4 questions:

- Hypothesis class of functions: *What is the set of functions we allow the predictor to take?*
- Loss function for training: *How do we measure how well the predictor performs on the training data?*
- Regularisation: *How do we construct predictors from only training data that performs well on unseen test data?*
- Cross-validation: *What is the procedure for searching over the space of models?*

Hypothesis class of function

Assume N examples(data points) $x_n \in \mathbb{R}^D$ and the scalar labels $y_n \in \mathbb{R}^D$. In supervised learning algorithms, it can be set to pairs of $(x_1, y_1), \ldots, (x_N, y_N)$.

With the given data, a predictor $f(\cdot, \theta) : \mathbb{R}^D \to \mathbb{R}$ parametrised by θ.

We need to find a good parameter $\theta^\star$ that can fit:

$$f(x_n, \theta^\star) \approx y_n \quad \text{for all } n = 1, \ldots, N$$

use notation $\hat{y}_n = f(x_n, \theta^\star)$ to represent the output of the predictor.

> Least-squares regression
>
> When the label y_n is real-valued, a popular choice of function class for predictors is *affine functions (linear functions)*.
>
> The linear predictor is
>
> $$f(x_n, \theta) = \theta_0 + \sum_{d=1}^{D} \theta_d x_n^{(d)}$$
>
> which takes x_n as input and produces a real-valued output: $f : \mathbb{R}^D \rightarrow \mathbb{R}$

Loss function for training

Loss function $l(y_n, \hat{y}_n)$ is used to define how well the predictor fits the data.

It tasks the *ground truth label* (y_n) and *the prediction* $(\hat{y}_n)$ as input and produces a non-negative number (loss) representing how much error made on the prediction.

The goal is to minimise the average loss on the training dataset N for finding a good parameter vector $\theta^\star$.

The set of data points is *independent* and *identically distributed*.

> Which means any 2 data points do not statistically depend on each other, meaning that the empirical mean is a good estimate of the population mean.
>
> This implies that we can use the empirical mean of the loss on the training data.

Consider a training set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, data points (example) matrix $X := [x_1, \ldots, x_N]^T \in \mathbb{R}^{N \times D}$, and label vector $y := [y_1, \ldots, y_N]^T \in \mathbb{R}^N$, then the average loss is given by

$$R_{emp}(f, X, y) = \frac{1}{N} \sum_{n=1}^{N} l(y_n, \hat{y}_n),$$

where $\hat{y}_n = f(x_n, \theta)$.

This equation is called the *empirical risk* and this general strategy for learning is called *empirical risk minimisation.*

> Least-Square Loss
>
> Specify the measure of the cost as the squared loss $l(y_n, \hat{y}_n) = (y_n - \hat{y_n})^2$
>
> To minimise the empirical risk
>
> $$min_{\theta \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} \ (y_n - f(x_n, \theta))^2$$
>
> where the predictor $\hat{y}_n = f(x_n, \theta) = \theta^T x_n,$
>
> $$min_{\theta \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^{N} \ (y_n - \theta^T x_n)^2$$
>
> This equation can be equivalently expressed in matrix form
>
> $$min_{\theta \in \mathbb{R}^D} \frac{1}{N} ||y - X\theta||^2$$
>
> known as the *least-squares problem.*

The predictor should perform well on unseen test data set, or more formally, minimise the expected risk.

$$R_{\text{true}}(f) = \mathbb{E}_{x,y}[l(y, f(x))]$$

The notation $R_{\text{true}}(f)$ indicates that this is the true risk if we had access to an infinite amount of data.

The expectation $\mathbb{E}$ is over the infinite set of all possible data and labels.

Machine learning applications have different types of performance measure.

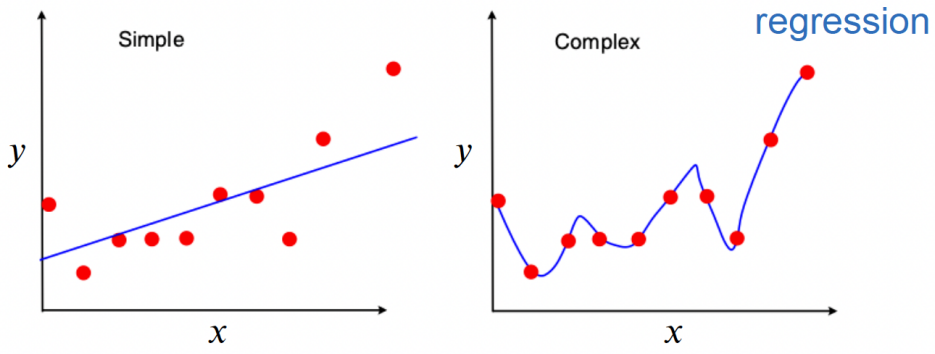
- For classification: accuracy, AUC, F1 score, etc.
- For detection: mean average precision, mIoU, etc.
- For image de-noise/super resolution: SSIM, PSNR, etc.

Regularisation to reduce overfitting

Test sets: the unseen data by holding out a proportion of the whole dataset.

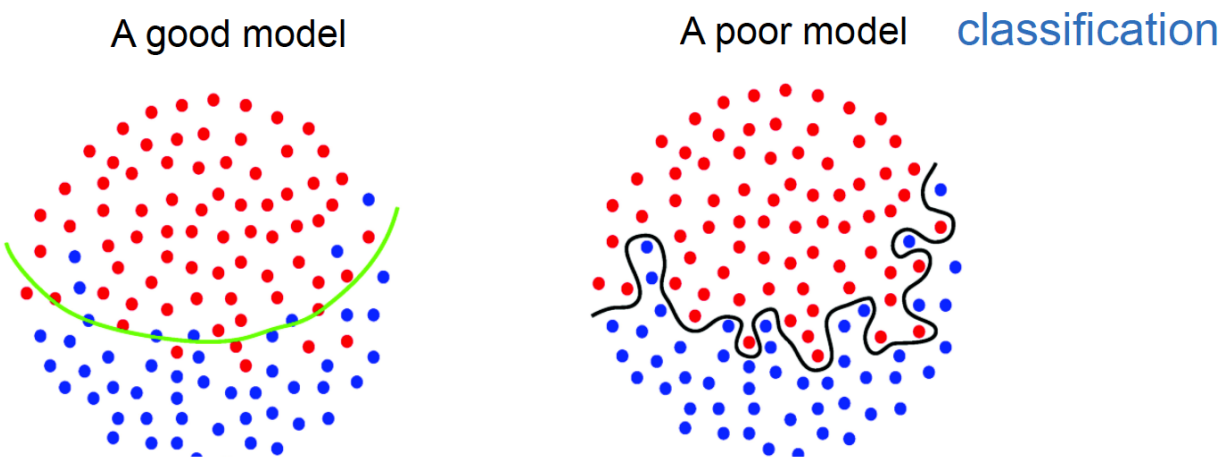
Split the finite data set into a training (to fit the model) and a test (to evaluate generalisation performance) set.

Empirical risk minimisation can lead to overfitting, having very small average loss on training set but large average loss on test set



This simple model fits the training data less well.

This complex model fits the training data very well.



(the test risk $R_{emp}(f, X_{test}, y_{test})$ is much larger than training risk $R_{emp}(f, X_{train}, y_{train})$)

, which tends to occur when we have little data and a complex hypothesis class.

Regularisation is a way to compromise between accurate solution of empirical risk minimisation and the size or complexity of the solution.

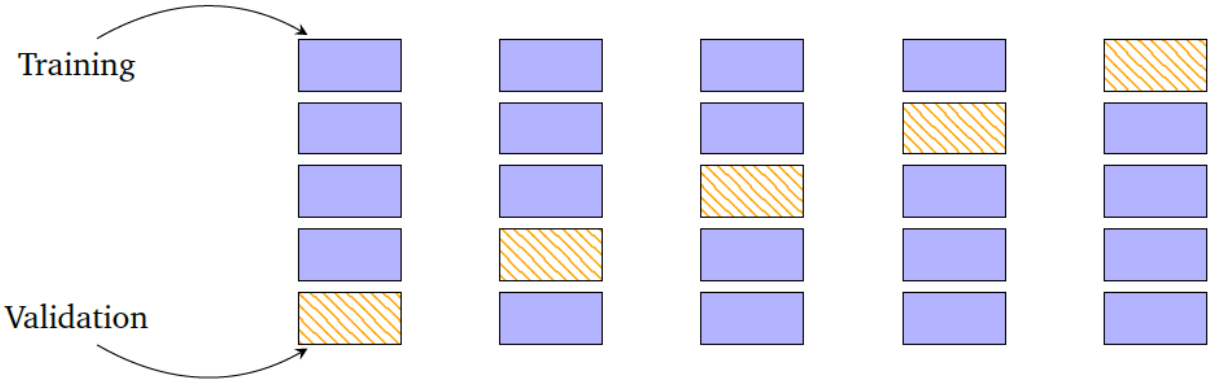
> Regularised Least Squares
>
> Regularisation discourages complex or extreme solutions to an optimisation problem.
>
> Adding ‘regularised’ problem to previous example is to add a penalty term:
>
> $$min_{\theta \in \mathbb{R}^D} \frac{1}{N} \|y - X\theta\|^2 + \lambda \|\theta\|^2$$
>
> The term $\|\theta\|^2$ is called *regulariser,* the parameter λ is called *regularisation parameter/penalty term.*

Cross-Validation to assess the generalisation performance

Validation set: subset of training data set, the amount is limited, and use as much of data available to train the model.

K-fold cross-validation effectively partitions the data into K chunks.

Partition the dataset into 2 sets $D = R \cup V$, so they don't overlap, where R is training set, V is validation set.



After training, access the performance of predictor f on validation set V by computing Root Mean Square Error. Then cycle through all possible partitioning of validation and training sets, and compute the average generalisation error

$$\mathbb{E}_V[R(f, V)] \approx \frac{1}{K} \sum_{k=1}^{K} R(f^{(k)}, V^{(k)})$$

where $R(f^{(k)}, V^{(k)})$ is the risk (RMSE) on validation set $V^{(k)}$ for predictor $f^{(k)}$.

The approximation has two sources:

- first, due to the finite training set, which results in not the best possible $f^{(k)}$;
- and second, due to the finite validation set, which results in an inaccurate estimation of the risk $R(f^{(k)}, V^{(k)})$.

A potential disadvantage of K-fold cross-validation is the computational cost of training the model K times.